



Functional Architecture

Brian Lonsdorf
@drboolean



<https://bit.ly/2Xq8CU4>

Goal?

- Modular
- Extendable
- Performant
- Maintainable
- Readable
- etc



```

    el.innerHTML = id + ' <b> ' + ms + ' ms</b><br />' + el.innerHTML;
    el.scrollTop = 0;
}

t = Date.now();

container.innerHTML = '';

this.showTime( container.id + ' clear ', Date.now() - t );

for( let i = 0; i < data.length; i++ )
{
    container.appendChild( this.createItem( data[i] ) );
}

window.scrollTo( 0, 0 );

this.showTime( container.id + ' populate', Date.now() - t );

id = item.getAttribute( 'data-id' );
has = _.filter( container.children, e => e.getAttribute( 'data-id' ) === id )
    .length !== 0;

if( !has )
{
    const clone = item.cloneNode( true );

    clone.classList.remove( 'active' );
    clone.getElementsByTagName( 'span' )[0].innerHTML = '';

    container.insertBefore( clone, container.firstChild );
}

item = document.createElement( 'div' );

item.innerHTML = `<span>${ data.id } : ${ data.distance.toFixed( 4 ) }</span>`;
item.classList.add( 'item' );
item.style.width = `${ data.canvas.width }px`;
item.style.height = `${ data.canvas.height }px`;
item.setAttribute( 'data-id', data.id );
item.setAttribute( 'data-original-id', data.originalId );

for( const rect of data.nodes )
{
    const node = document.createElement( 'div' );
    const style = `top:${ rect.y }px; left:${ rect.x }px; width:${
        | rect.width
    }px; height:${ rect.height }px`;

    node.innerHTML = rect.id;
    node.setAttribute( 'style', style );

    item.appendChild( node );
}

```

```
el.innerHTML = id + ' : <b> + ms + 'ms</b><br />' + el.innerHTML;
el.scrollTop = 0;
}
```

```
t = Date.now();

container.innerHTML = '';

this.showTime( container.id + ' clear ', Date.now() - t );

for( let i = 0; i < data.length; i++ )
{
  container.appendChild( this.createItem( data[i] ) );
}

window.scrollTo( 0, 0 );

this.showTime( container.id + ' populate', Date.now() - t );
```

```
id = item.getAttribute( 'data-id' );
has = _.filter( container.children, e => e.getAttribute( 'data-id' ) === id )
  .length !== 0;

if( !has )
{
  const clone = item.cloneNode( true );

  clone.classList.remove( 'active' );
  clone.getElementsByTagName( 'span' )[0].innerHTML = '';

  container.insertBefore( clone, container.firstChild );
}

item = document.createElement( 'div' );
```

```
item.innerHTML = `<span>${ data.id } : ${ data.distance.toFixed( 4 )}</span>`;
item.classList.add( 'item' );
item.style.width = `${ data.canvas.width }px`;
item.style.height = `${ data.canvas.height }px`;
item.setAttribute( 'data-id', data.id );
item.setAttribute( 'data-original-id', data.originalId );

for( const rect of data.nodes )
{
  const node = document.createElement( 'div' );
  const style = `top:${ rect.y }px; left:${ rect.x }px; width:${
    | rect.width
  }px; height:${ rect.height }px`;

  node.innerHTML = rect.id;
  node.setAttribute( 'style', style );

  item.appendChild( node );
}
```

```
el.innerHTML = id + ' : <b> ' + ms + ' ms</b><br />' + el.innerHTML;
el.scrollTop = 0;
}
```

```
t = Date.now();

container.innerHTML = '';

this.showTime( container.id + ' clear ', Date.now() - t );

for( let i = 0; i < data.length; i++ )
{
  container.appendChild( this.createItem( data[i] ) );
}

window.scrollTo( 0, 0 );

this.showTime( container.id + ' populate', Date.now() - t );
```

```
id = item.getAttribute( 'data-id' );
has = _.filter( container.children, e => e.getAttribute( 'data-id' ) === id )
      .length !== 0;

if( !has )
{
  const clone = item.cloneNode( true );

  clone.classList.remove( 'active' );
  clone.getElementsByTagName( 'span' )[0].innerHTML = '';

  container.insertBefore( clone, container.firstChild );
}

item = document.createElement( 'div' );
```

```
item.innerHTML = `<span>${ data.id } : ${ data.distance.toFixed( 4 )}</span>`;
item.classList.add( 'item' );
item.style.width = `${ data.canvas.width }px`;
item.style.height = `${ data.canvas.height }px`;
item.setAttribute( 'data-id', data.id );
item.setAttribute( 'data-original-id', data.originalId );

for( const rect of data.nodes )
{
  const node = document.createElement( 'div' );
  const style = `top:${ rect.y }px; left:${ rect.x }px; width:${
    | rect.width
  }px; height:${ rect.height }px`;

  node.innerHTML = rect.id;
  node.setAttribute( 'style', style );

  item.appendChild( node );
}
```

populateContainer

makeItemActive

renderPattern

fullName() {}
averageScore() {}
friends() {}
lastLogin() {}

toString() {}
getCountry() {}
getState() {}
getNumber() {}
getStreet() {}

validate() {}
isPresent() {}
isUnique() {}
inEnum() {}

expandZipCode() {}
getLatLng() {}
getPolygon() {}

save() {}
find() {}
destroy() {}

dedupe() {}
findMinimum() {}
intersection() {}

User

```
fullName() {}  
averageScore() {}  
friends() {}  
lastLogin() {}
```

Address

```
toString() {}  
getCountry() {}  
getState() {}  
getNumber() {}  
getStreet() {}
```

Validations

```
validate() {}  
isPresent() {}  
isUnique() {}  
inEnum() {}
```

Geolocate

```
expandZipCode() {}  
getLatLng() {}  
getPolygon() {}
```

Repo

```
save() {}  
find() {}  
destroy() {}
```

Utils

```
dedupe() {}  
findMinimum() {}  
intersection() {}
```

```
{street, number, state, county, country, zipcode }
```

```
{firstName, lastName, email, username, address, score, friends }
```

```
{ backgroundColor, amountPerPage, expanded, resolution }
```



```
{street, number, state, county, country, zipcode }
```

Address

```
{firstName, lastName, email, username, address, score, friends }
```

User

```
{ backgroundColor, amountPerPage, expanded, resolution }
```

Preferences

modified

file



Copyright © 2017



Domain-Driven

DESIGN

Tackling Complexity in the Heart of Software



Eric Evans

Foreword by Martin Fowler

Copyright © 2017

Metaphors?

```
save(name, callback) {
  const todo = this.createTodo(name)
  if(name.trim()) {
    if(find(this.todos, t => t.name == todo.name)) {
      return callback("duplicate todo")
    } else {
      this.todos.unshift(todo)
      this.setTodos(callback)
    }
  } else {
    return callback("name can't be empty")
  }
}
```

```
toggle(todo, callback) {
  todo.toggleComplete()
  this.setTodos(callback)
}
```

- Know about context and domain
- Mixed metaphors: Processor, Converter, etc
- Evolves/Blurs over time
- Hodge-podge of functionality in each object

Procedures

```
save(name, callback) {
  const todo = this.createTodo(name)
  if(name.trim()) {
    if(find(this.todos, t => t.name == todo.name)) {
      return callback("duplicate todo")
    } else {
      this.todos.unshift(todo)
      this.setTodos(callback)
    }
  } else {
    return callback("name can't be empty")
  }
}

toggle(todo, callback) {
  todo.toggleComplete()
  this.setTodos(callback)
}
```

```
save(name, callback) {
  const todo = this.createTodo(name)
  if(name.trim()) {
    if(find(this.todos, t => t.name == todo.name)) {
      return callback("duplicate todo")
    } else {
      this.todos.unshift(todo)
      this.setTodos(callback)
    }
  } else {
    return callback("name can't be empty")
  }
}

toggle(todo, callback) {
  todo.toggleComplete()
  this.setTodos(callback)
}
```

```
save(name, callback) {
  const todo = this.createTodo(name)
  if(name.trim()) {
    if(find(this.todos, t => t.name == todo.name)) {
      return callback("duplicate todo")
    } else {
      this.todos.unshift(todo)
      this.setTodos(callback)
    }
  } else {
    return callback("name can't be empty")
  }
}

toggle(todo, callback) {
  todo.toggleComplete()
  this.setTodos(callback)
}
```

Procedures

```
save(name, callback) {
  const todo = this.createTodo(name)
  if(name.trim()) {
    if(find(this.todos, t => t.name == todo.name)) {
      return callback("duplicate todo")
    } else {
      this.todos.unshift(todo)
      this.setTodos(callback)
    }
  } else {
    return callback("name can't be empty")
  }
}

toggle(todo, callback) {
  todo.toggleComplete()
  this.setTodos(callback)
}
```

```
save(name, callback) {
  const todo = this.createTodo(name)
  if(name.trim()) {
    if(find(this.todos, t => t.name == todo.name)) {
      return callback("duplicate todo")
    } else {
      this.todos.unshift(todo)
      this.setTodos(callback)
    }
  } else {
    return callback("name can't be empty")
  }
}

toggle(todo, callback) {
  todo.toggleComplete()
  this.setTodos(callback)
}
```

- Can i run this twice in a row?
- Which order do I need to run these in?
- Is it changing other parts of the program?
- How does it interact with others?

```
save(name, callback) {
  const todo = this.createTodo(name)
  if(name.trim()) {
    if(find(this.todos, t => t.name == todo.name)) {
      return callback("duplicate todo")
    } else {
      this.todos.unshift(todo)
      this.setTodos(callback)
    }
  } else {
    return callback("name can't be empty")
  }
}

toggle(todo, callback) {
  todo.toggleComplete()
  this.setTodos(callback)
}
```

```
// associative
```

```
add(add(x, y), z) == add(x, add(y, z))
```

```
// commutative
```

```
add(x, y) == add(y, x)
```

```
// identity
```

```
add(x, 0) == x
```

```
// distributive
```

```
add(multiply(x, y), multiply(x, z)) == multiply(x, add(y,z))
```

Functions with defined contracts


```
class User {
  constructor(firstName, lastName) {
    this.firstName = firstName
    this.lastName = lastName
  }

  fullName() {
    return this.firstName + ' ' + this.lastName
  }
}
```

```
const user = new User('Bobby', 'Fischer')
user.fullName()
// Bobby Fischer
```

```
const user = {firstName: 'Bobby', lastName: 'Fischer'}
const fullName = (firstName, lastName) => [firstName, lastName].join(' ')

fullName(user.firstName, user.lastName)
// Bobby Fischer
```

```
const user = {firstName: 'Bobby', lastName: 'Fischer'}  
const joinWithSpace = (...args) => args.join(' ')
```

```
joinWithSpace(user.firstName, user.lastName)
```

```
joinWithSpace('a', 'b', 'c') // 'a b c'
```

```
joinWithSpace(joinWithSpace('a', 'b'), 'c') // 'a b c'
```

```
joinWithSpace('a', joinWithSpace('b', 'c')) // 'a b c'
```

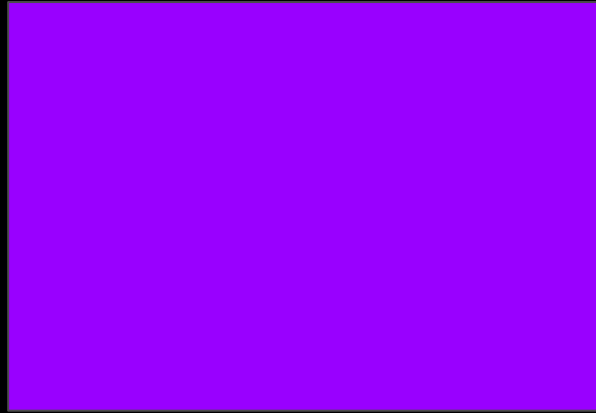
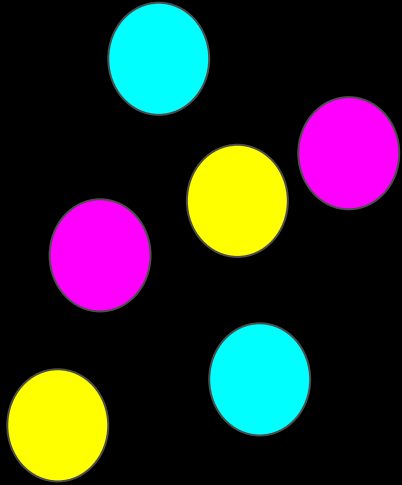
```
joinWithSpaces = joinable => joinable.join(' ')
```

```
joinWithSpaces([user.firstName, user.lastName])
```

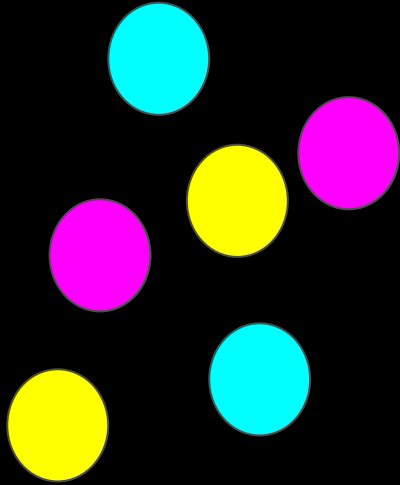
```
const identity = a => a
```

Highly
generalized
functions

Composition



Composition

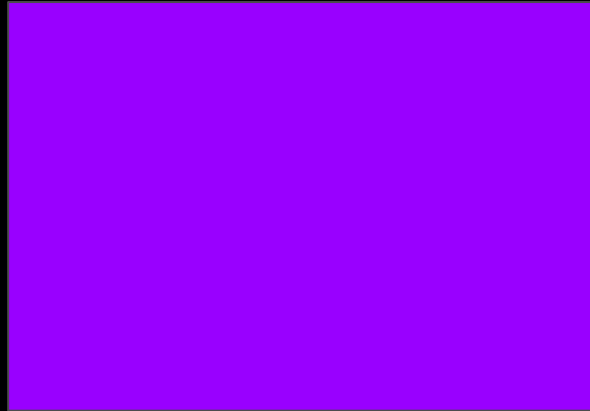


- Satisfy use cases
 - Simple, understandable pieces
 - Reuse
-
- Harder to change implementation
 - Harder for user to compose

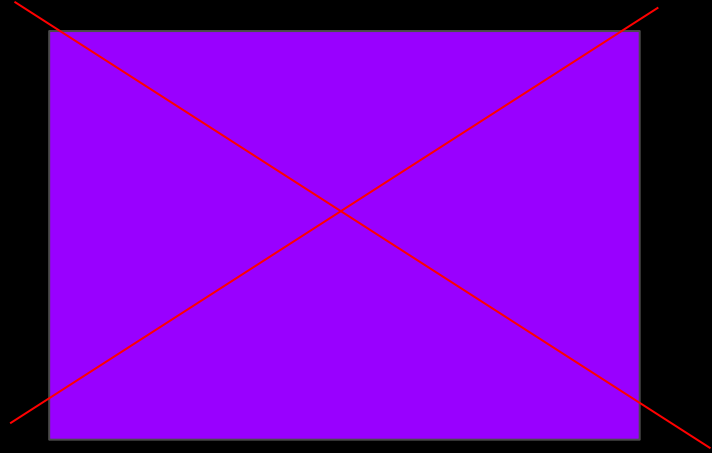
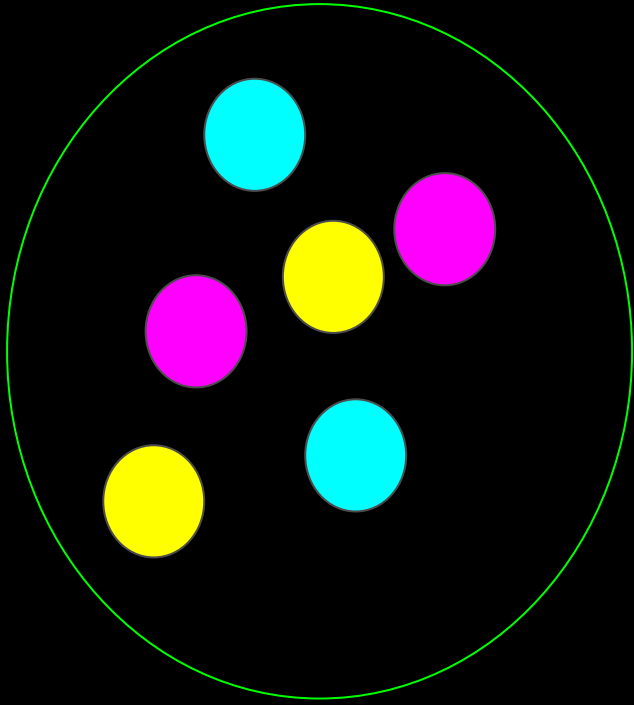
Composition

- Flexibility in implementation changes
- Less use cases to support

- Flags, if/else
- Won't satisfy all cases
- Less reuse



Composition



Definition [\[edit \]](#)

A group is a **set**, G , together with an **operation** \cdot (called the *group law* of G) that combines any two **elements** a and b to form another element, denoted $a \cdot b$ or ab . To qualify as a group, the set and operation, (G, \cdot) , must satisfy four requirements known as the *group axioms*:^[5]

Closure

For all a, b in G , the result of the operation, $a \cdot b$, is also in G .^[b]

Associativity

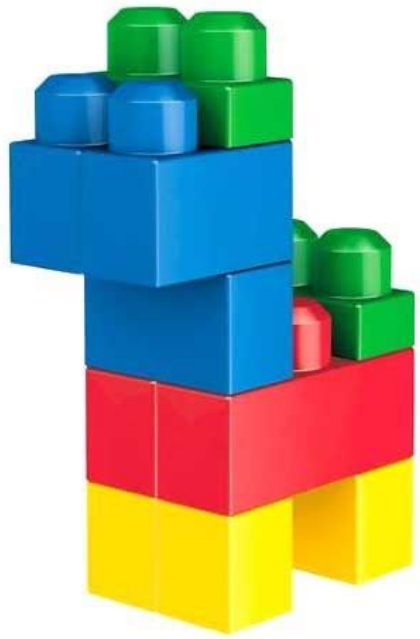
For all a, b and c in G , $(a \cdot b) \cdot c = a \cdot (b \cdot c)$.

Identity element

There exists an element e in G such that, for every element a in G , the equation $e \cdot a = a \cdot e = a$ holds. Such an element is unique element.

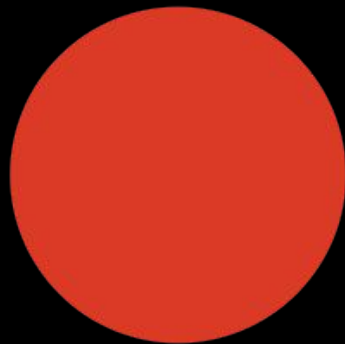
Inverse element

For each a in G , there exists an element b in G , commonly denoted a^{-1} (or $-a$, if the operation is denoted "+"), such that $a \cdot b = b \cdot a = e$.





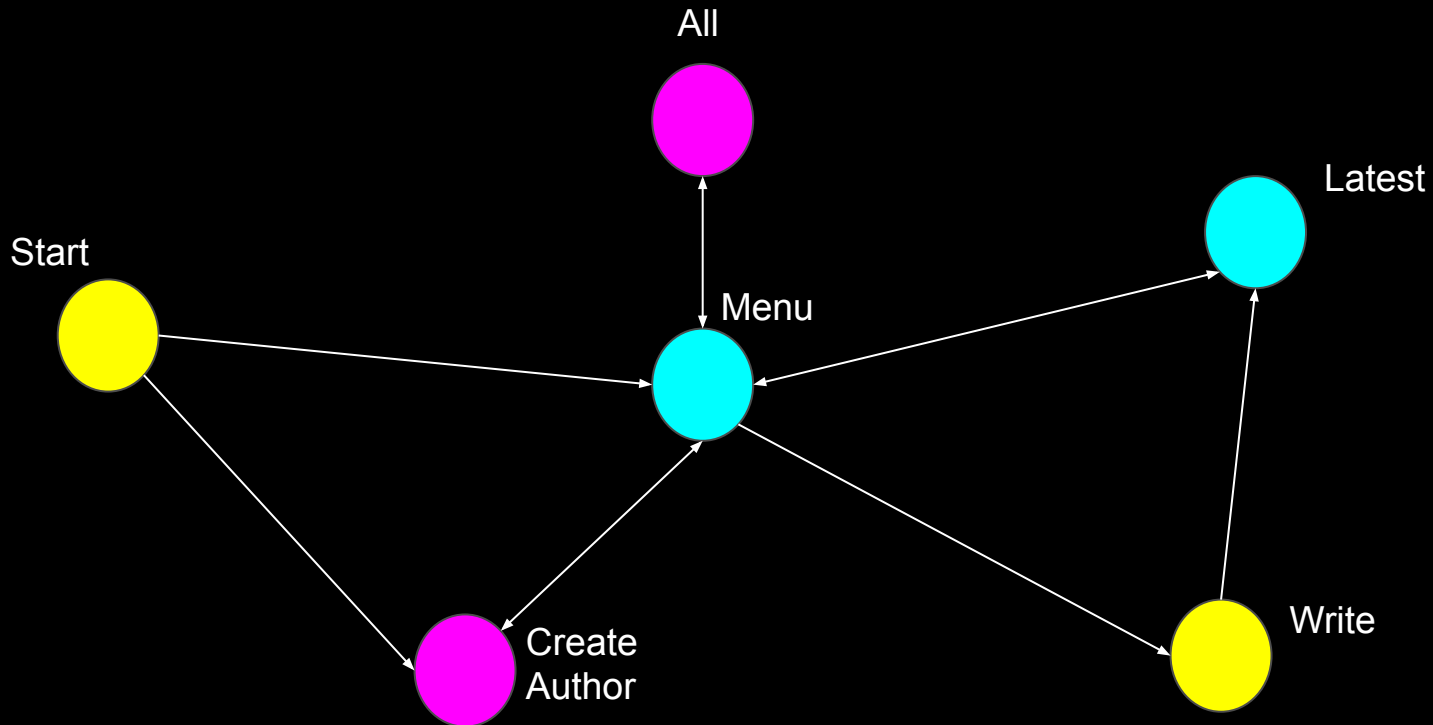
Favor
composable
functions,
mostly







CLI blog



Exercises

- Monoid: <https://codepen.io/drboolean/pen/MpKpee>
- Modelling Fns: <https://codepen.io/drboolean/pen/YZwrGK>
- Monad Transformers: <https://codepen.io/drboolean/pen/NQKByP>

Code

<https://drive.google.com/file/d/1XQr5SZvTJ7dFYChr5kV42zO2FMhSATOa/view?usp=sharing>